ADVANCES IN KNOWLEDGE-BASED SOFTWARE ENGINEERING

Walt Truszkowski
Head, Automation Technology Section
Code 522.3
Goddard Space Flight Center
Greenbelt, MD 20771

ABSTRACT

The underlying hypothesis of the work reported on in this paper is that a rigorous and compre- hensive software reuse methodology can bring about a more effective and efficient utilization of constrained resources in the development of large-scale software systems by both the Government and industry. It is also believed that correct use of this type of software engineering methodology can significantly contribute to the higher levels of reliability that will be required of future operational systems.

This paper presents an overview and discussion of current research in the development and application of two systems that support a rigorous reuse paradigm: the Knowledge-Based Software Engineering Environment (KBSEE) and the Knowledge Acquisition for the Preservation of Tradeoffs and Underlying Rationales (KAPTUR) systems. The paper concentrates on a presentation of operational scenarios which highlight the major functional capabilities of the two systems.

THE KNOWLEDGE-BASED SOFTWARE ENGINEERING ENVIRONMENT (KBSEE)

The KBSEE (refs.1, 2) is one of the systems that is being used to substantiate the hypothesis stated above. This system currently supports a comprehensive software specification reuse capability. A central concept for the KBSEE is the concept of a domain. A domain, in our context, is any class of related objects. These objects can be as diverse as cruise control systems for cars, elevator systems, or spacecraft command and control systems (in fact all three of these domains have been used to demonstrate the functionality of the KBSEE system to date). A high-level view of an end-to-end (reusable object definition to target system specification) KBSEE scenario is as follows. A domain model is developed as part of an initial exercise in populating the reuse knowledge base and, in an internal standard form, is stored in the KBSEE's reuse repository. When the requirement for a new instance, or target system/application, of that model or portion thereof arises, the KBSEE provides the software engineer with the capabilities to tailor the existing model to meet the target application's requirements and constraints. The output from the current system is the specification for the required target system. If, in the development of the specifications for the new target system, some of the requirements or constraints cannot be met by the current domain model a feedback to the domain modeling capabilities of the KBSEE allows the current model to be modified in order that the new requirements or constraints may be reflected. This process thus allows the domain model to evolve to a richer model. This phenomenon gives rise to the concept of an evolutionary domain life cycle. The Figure 1. graphically illustrates the major processes supporting the domain modeling and target system generation stages of the KBSEE. Model-wise the KBSEE is highly object-oriented and there is a similarity in concept between the tailoring of a domain model and the use of class definitions in arriving at instances of objects.

A major experiment has been conducted to better explore the applicability of the KBSEE in support of the development of specifications for spacecraft control center software components. Based on an extensive domain analysis of several control center software systems a generic control center software system model was established. This model was used as the basis for the definition of reusable software specifications for target applications in the control center environment. As depicted in the following two figures the KBSEE provides two basic types of capabilities: creation of a domain specification and generation of target system specifications.

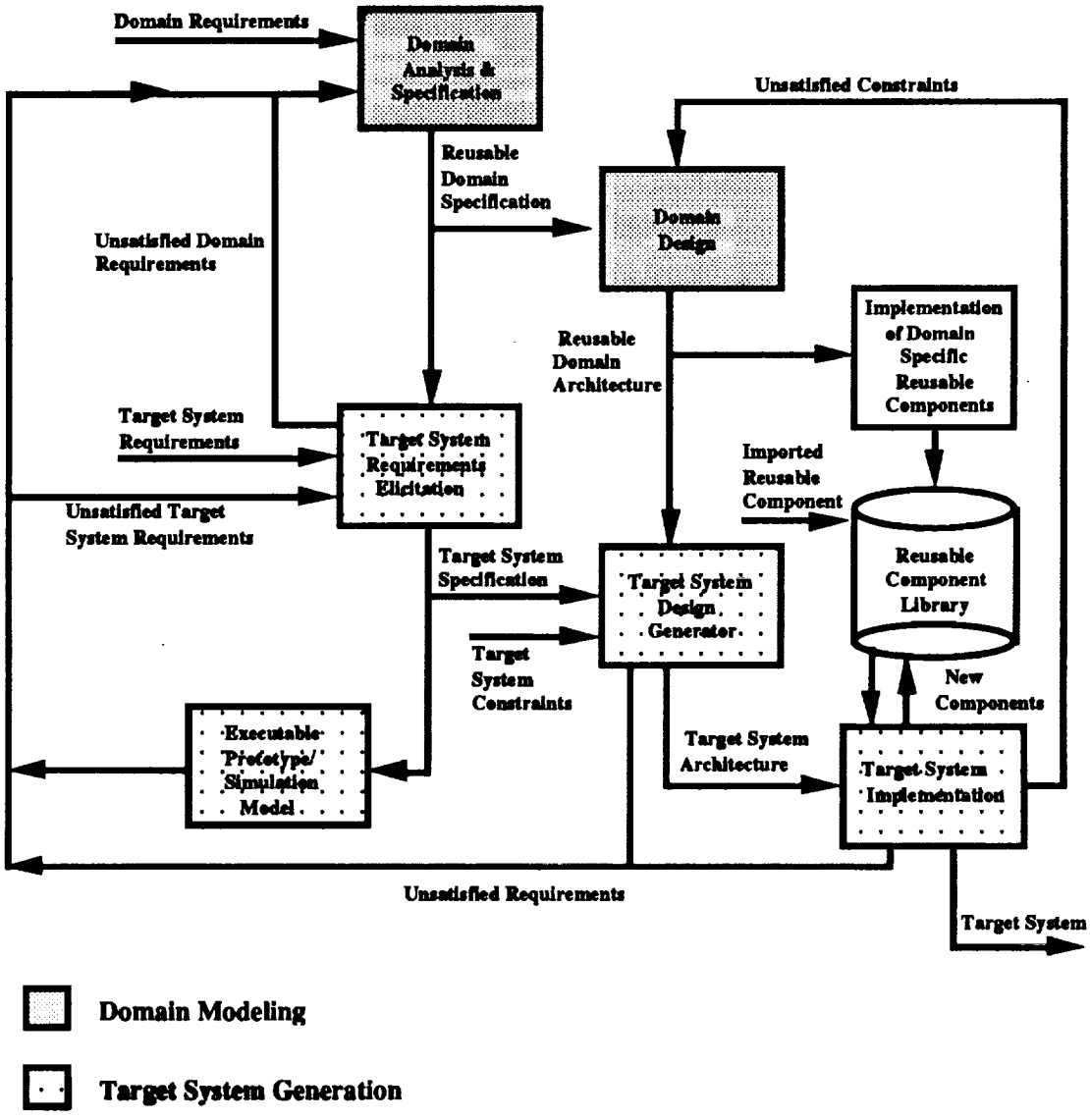


Figure 1. The Evolutionary Domain Life Cycle Model

The Figure 2. illustrates the object-oriented domain model specification and reusable component generation capabilities provided by the KBSEE. In the development of reusable objects in a domain of interest the software engineer is provided capability by the KBSEE to develop domain component specifications in a number of representations, namely:

1. object communication diagrams which are hierarchically structured and show how objects communicate with each other through the mechanism of message passing,
2. aggregation hierarchies which supports the decomposition of complex aggregate objects into less complex objects eventually leading to simple objects at the leaves of the hierarchy,
3. generalization/specialization hierarchies which support the IS-A relationship and inheritance relationships between classes and instance objects, and
4. state transition diagrams which reflect the fact that objects are sequential processes which may be represented by finite state machines and documented by state transitions.

Once the specifications have been developed by the software engineer the multiple representations are checked for consistency among themselves. If there are any inconsistencies these are brought to the attention of the software engineer for corrective action. Once all the inputs are consistent they are transformed into an internal format for storage in and later retrieval from the domain object repository.

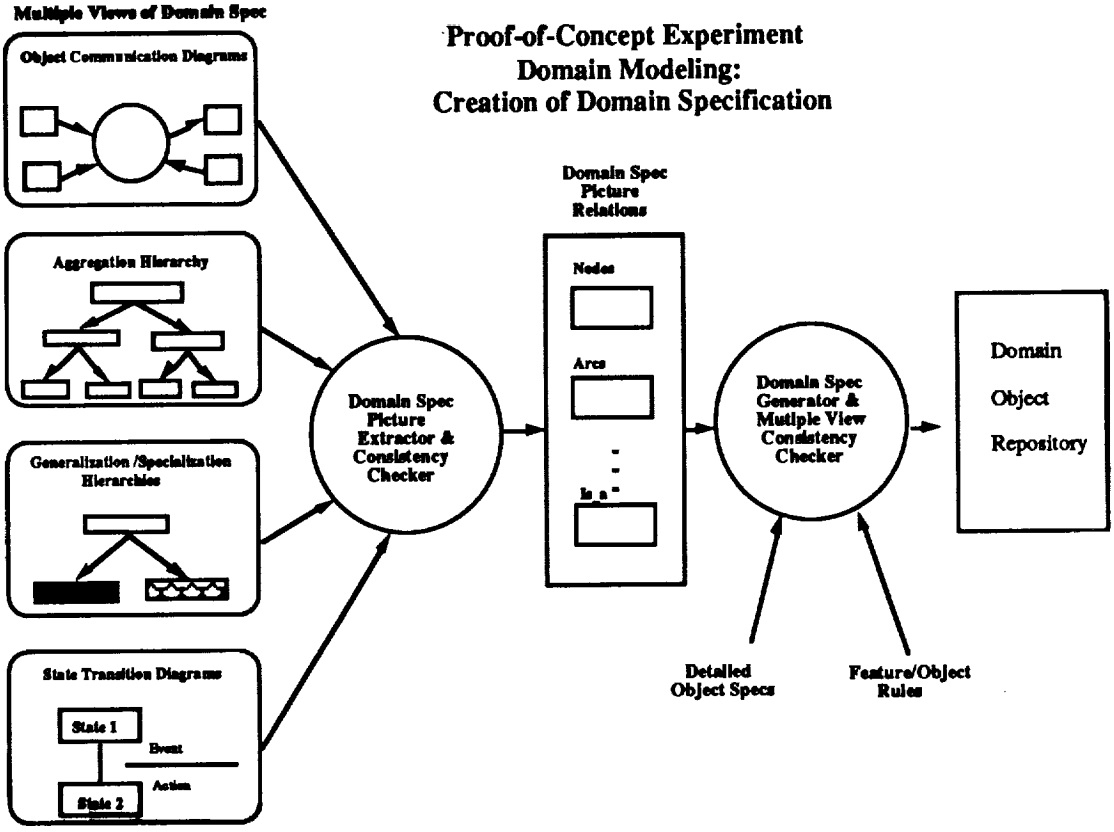


Figure 2. Creation of a Domain Specification and Reusable Components

The following Figure 3. depicts the major steps in the process of generating target system specifications using the KBSEE. In many ways it is the reverse process of the domain modeling processes depicted in Figure 2. A typical scenario for this aspect of the KBSEE would be as follows. A software engineer would invoke the KBSEE with a specific target system in mind. The KBSEE's knowledge elicitation component, KBRET, would engage the software engineer in an interactive exchange during which time the requirements and constraints for the target system would be made known to the KBSEE. This process currently involves the engineer selecting from requirements and constraints currently supported by the appropriate domain model in the KBSEE knowledge base of reusable objects. If the target system requires a feature not currently satisfied by the domain model the KBSEE, through a feedback mechanism, allows the feature to be added to the domain model through the processes depicted in Figure 2. Once all of the target system objects have been defined the KBSEE, through its target system picture generator, creates the multiple views of the target system specification which is the current output of the system.

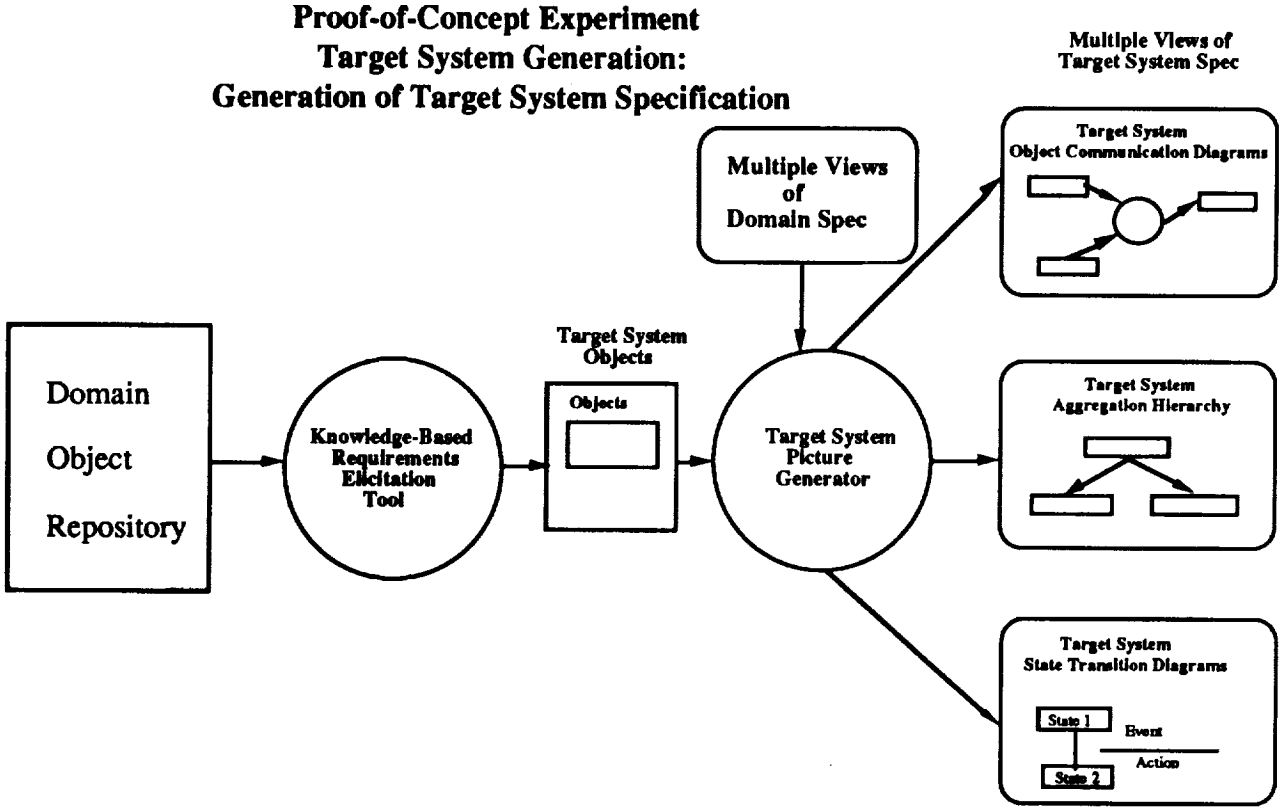
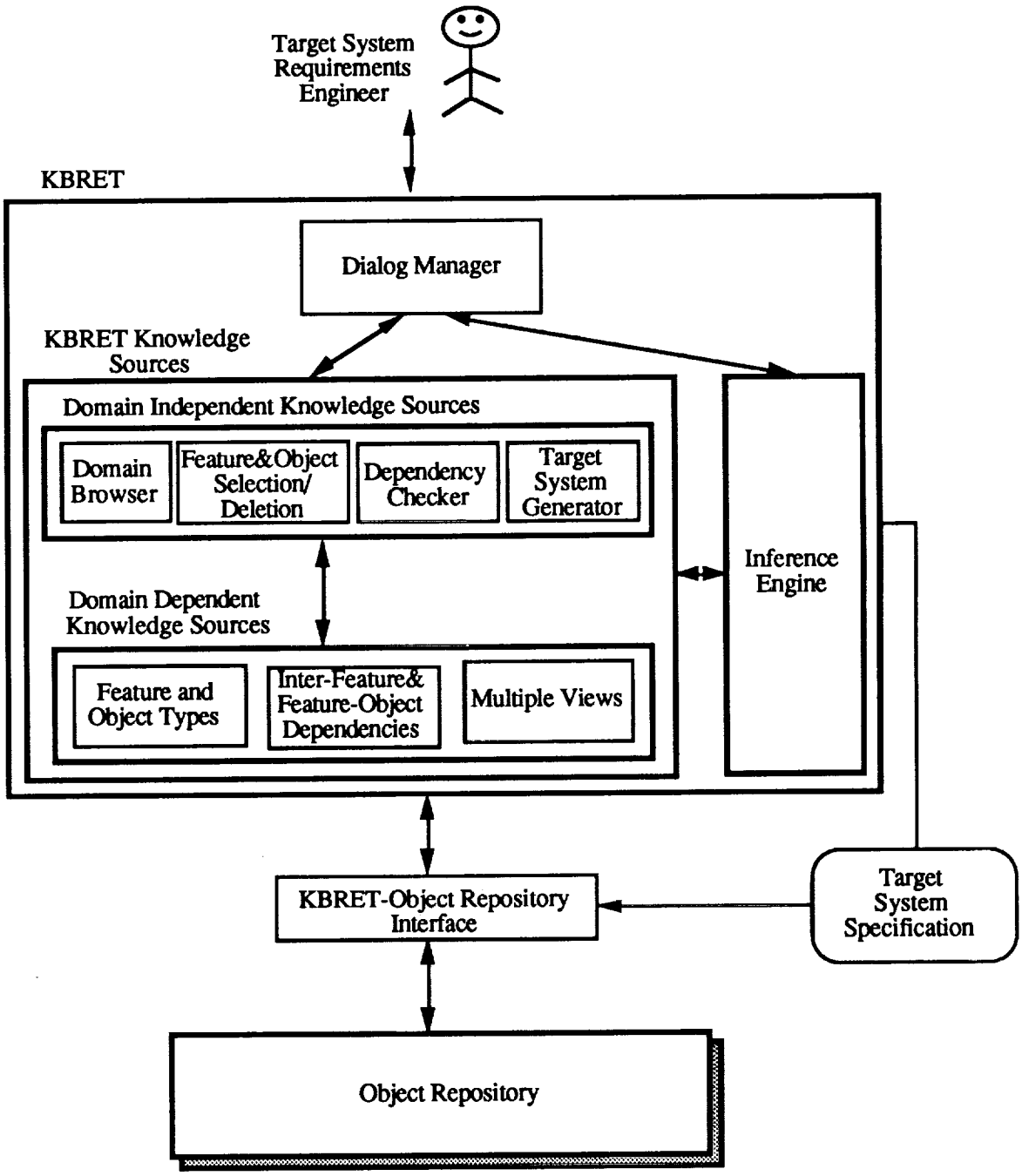


Figure 3. Use of the KBSEE in Target System Specification Generation

As mentioned above, a major innovative feature in the KBSEE's support for target system specification is depicted in the above figure in the left-most circle. It is the Knowledge-Based Requirements Elicitation Tool (KBRET) and a fuller discussion of this tool follows. This tool supports the development of target system specifications by entering into an interactive session with the target system designer. During the session KBRET elicits target system requirements by having the designer select desired features and object types associated with the domain model. KBRET queries the object repository to obtain the knowledge required for its reasoning in support of the target system specification generation process. Modern browsing techniques are being used by KBRET to support this access.The Figure 4. illustrates the major components of the KBRET tool. A major attempt has been made to ensure that the user's interface to the KBSEE through the KBRET tool is as user accommodating as possible. An interesting and powerful aspect of the KBRET architecture is the organization of its knowledge base. The KBRET knowledge base is divided into domain dependent and domain independent portions. The domain independent portion support such activities as browsing, the selection and/or deletion of target system features by the user, a check for dependencies among selected features based on the domain model, and finally an invocation of the target system generator process. All of these activities apply equally to any domain selected. The domain dependent portion makes extensive use of the highly structures object-oriented reuse knowledge base and provides the data and information required to support the domain-specific specification development process.



Knowledge Based Requirements Elicitation Tool (KBRET)

Figure 4. Component View of the KBRET Tool

The present version of the KBSEE is implemented on a Sun and utilizes Software through Pictures as the multiple viewpoint graphical editor, Eiffel as the basis for the object repository, CLIPS for the KBRET knowledge elicitation component, and TAE+ for the KBSEE graphical interfaces.

KNOWLEDGE ACQUISITION FOR THE PRESERVATION OF TRADEOFFS AND UNDERLYING RATIONALES (KAPTUR)

The KAPTUR system (ref. 3) shares many of the goals as the KBSEE. The KAPTUR system is intended to support systematic reuse of knowledge and artifacts throughout the software development life-cycle. The main contribution of KAPTUR is the support it provides for the evaluation of potentially reusable artifacts, enabling the developer to make intelligent choices among the possibilities. KAPTUR is intended to provide as much information as possible, in an easily accessible form, to help clarify whether a given artifact is suitable for reuse in a given context.

KAPTUR is intended to preserve knowledge that is required or generated during the development process, but that is often lost because it is contextual, i.e., it does not appear directly in the end-products of the development process. Such knowledge includes issues that were raised during development, alternatives that were considered, and the reasons that were used to choose one alternative over another. Contextual information , in our sense, is usually only maintained as a memory in a developer's mind. As time passes, the memories become more vague and individuals become unavailable, and eventually the knowledge is lost. KAPTUR seeks to mitigate this process of information attrition by recording and organizing contextual knowledge as it is generated. From the lessons learned from previous development efforts, current developers can improve their insight into the problems at hand and their possible solutions.

The following Figure 5. illustrates the underlying operational philosophy of KAPTUR. Specifically it shows how KAPTUR would be used to explore alternative software architectures for a control center application processor. In the center of the diagram there is a knowledge base, called the Legacy, which contains information about the application domain. In our case the application domain is control center software systems. The information includes recommended architectures and information about previously developed systems. In the scenario depicted in Figure 5. the developer has available a set of software requirements and wants to begin defining an applications process to meet these requirements. The developer sits down at the KAPTUR workstation and issues a command whose meaning is something like the following: "I want to develop a control center applications process. Show me what they look like." In response the KAPTUR system displays the recommended generic architecture (upper right-hand box) as well as a stack of alternative architectures related to this requirement. Upon examining the recommended architecture the developer has the following options:

- examine the distinctive features of the recommended architecture,
- examine the alternatives by clicking on one of the windows behind the recommended architecture,
- define a new architecture,
- accept the recommended architecture.

The distinctive features of an architecture are those that are different from common practice or the recommended approach, or that represent a non-trivial decision about a significant issue. It is the prime purpose of KAPTUR to preserve the knowledge and analysis of the decisions associated with the distinctive features. Distinctive features may correspond to specific portions of an architecture (e.g., the interface between two subsystems) or they may represent some aspect of the architecture as a whole (e.g., the distribution of initialization functions to all subsystems of a system).

If the developer selects Distinctive Features, KAPTUR will list the distinctive features of the architecture being displayed., and will allow the developer to select one or more of these features. KAPTUR will then display a representation of the distinctive feature(s). In effect, the developer is afforded the opportunity to zoom into a view of a particular feature of an architecture. This is illustrated in the bottom-middle box in Figure 5.

The developer can then examine the Rationales for this feature, i.e., the reasoning underlying the decision that the feature represents. In the lower right-hand box in the Figure 5. the rationales are represented as a list of object-oriented design criteria that might underlie the decision. From this screen the developer can request even more detailed explanations by asking to view Trade-Offs that were considered in making the decision. The developer can also ask to view Alternatives to this decision, i.e., other systems that do not possess this feature because a different decision was made.

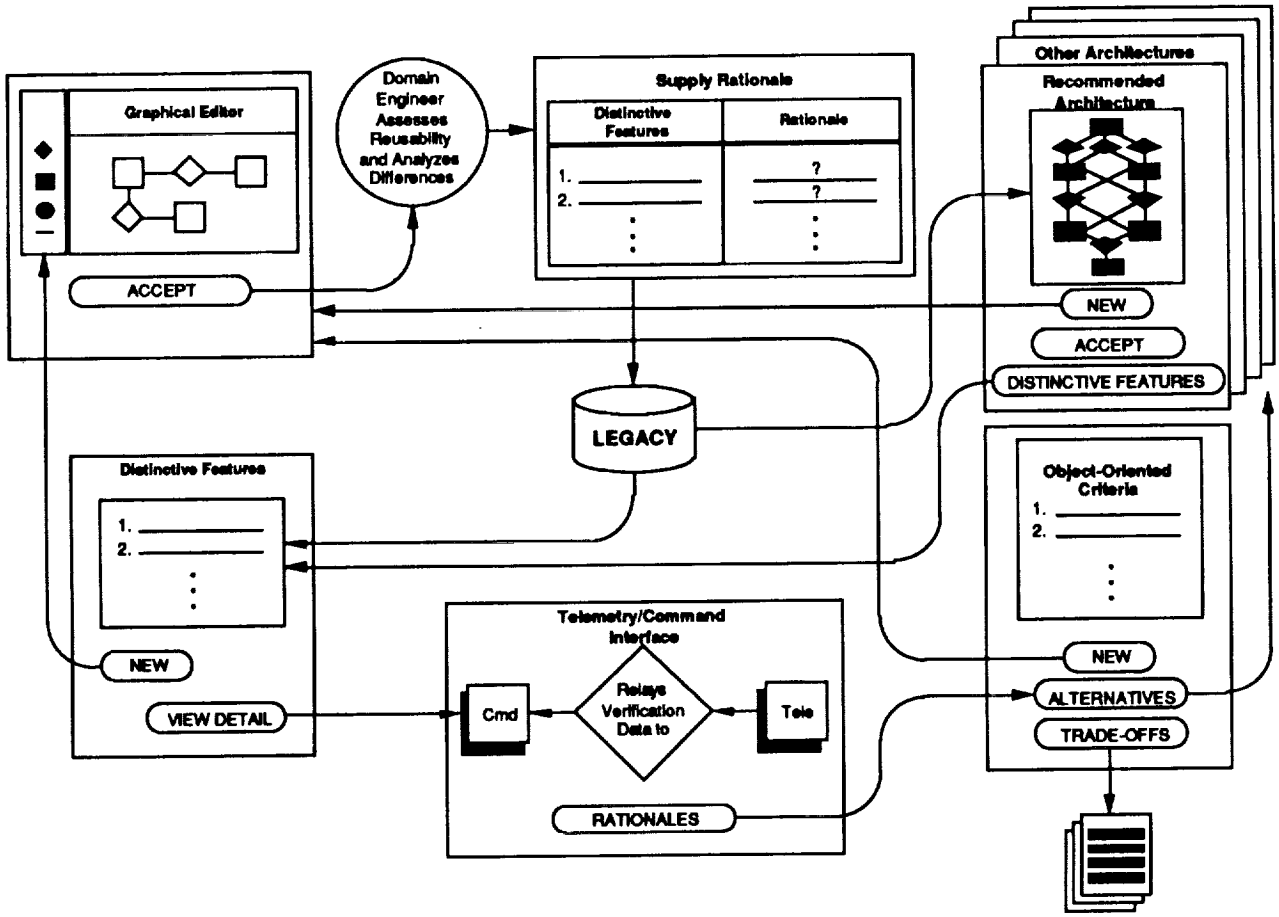


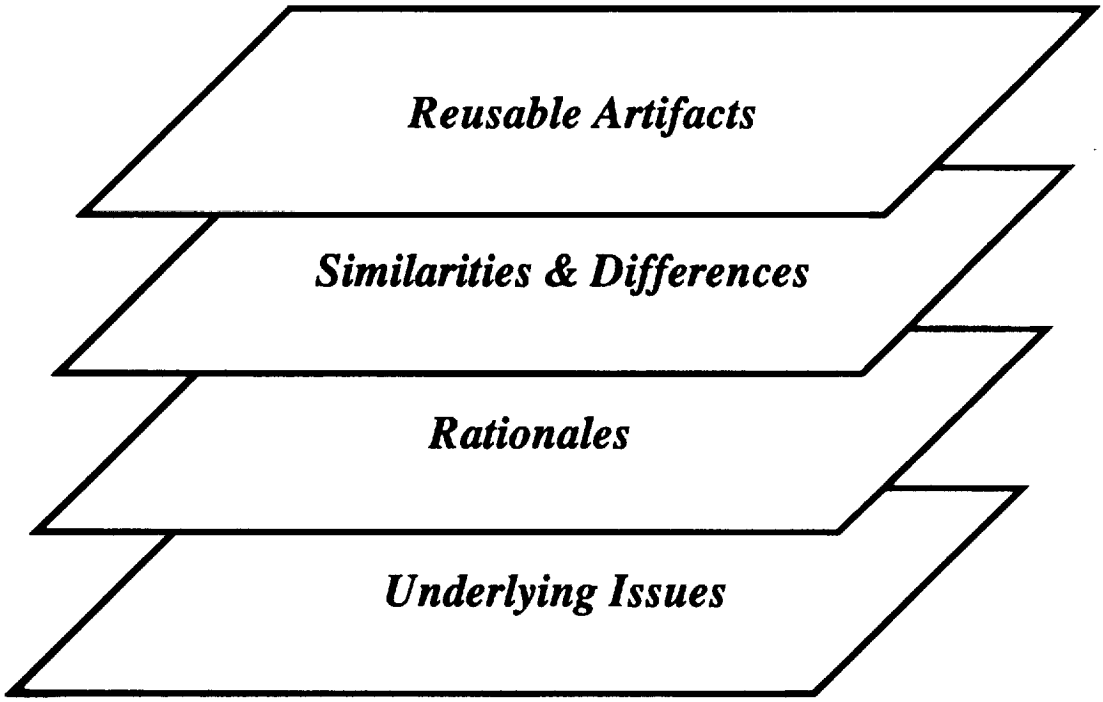
Figure 5. A Scenario View of the KAPTUR System

If the developer selects New (from either the Recommended Architecture or Alternative Architecture screens) a graphical editor will be invoked to allow the interactive definition of a new architecture. The definition of a new architecture need not start from scratch. A clipboard capability in KAPTUR allows the developer to select portions of the recommended and/or alternative architectures for inclusion into the new architecture.

Once a new architecture has been defined the developer will perform an analysis to determine the distinctive features of the new architecture, i.e., the ways in which it significantly differs from the recommended architecture. For each identified distinctive feature KAPTUR will prompt the developer to enter one or more rationales justifying the feature. This is shown in the top-middle box in Figure 5.

The new architecture, together with its rationales, then become part of the Legacy of the domain and will appear in the Alternatives list when KAPTUR is next used. This is how the evolution of domain requirements and solutions is captured in the knowledge base.

The knowledge in KAPTUR is stratified into four layers as depicted in Figure 6. It is through this multi-leveled knowledge base that KAPTUR is able to support the mechanisms for design knowledge capture within a robust reuse environment.



Layers in KAPTUR's Knowledge Base

Figure 6. Structure of the Underlying Knowledge Base for KAPTUR

SUMMARY AND CONCLUSIONS

The two systems described in this paper have been developed over a period of three years to serve as testbeds for the prototyping and evaluation of knowledge-based and advanced software engineering concepts needed to support a rigorous software reuse paradigm. Among the major concepts studied have been those associated with:

- representation of reusable software specifications
- consistency checking among various specification formalisms
- knowledge-based approaches for interactive requirements elicitation
- mechanisms for design knowledge capture
- hierarchical structuring of design knowledge
- knowledge-based browsing techniques
- user/system interaction
- object-oriented knowledge bases

Both of these systems are currently being used to focus on the issues associated with software reuse in the context of spacecraft control center software system specifications. As NASA missions become more complex, long-lived, and increasingly expensive the developmental and cost-savings benefits that can be derived from a well formulated reuse methodology take on added significance. Especially for those programs that have a long projected lifetime, the need for establishing and maintaining a "corporate memory" of reusable components and system development rationales becomes critical for an effective sustaining engineering activity. Over the next year both of these systems will be field-tested on real-time control center software development projects to help in further evaluating their effectiveness in operational settings.

We feel strongly that the concepts embodied in systems like the KBSEE and KAPTUR have application in any organization that is responsible for the timely and economic development of large software systems. Additionally, any organization responsible for the sustained engineering of large systems over a long period of time could profit from the design knowledge capture capabilities being investigated.

ACKNOWLEDGEMENTS

The KBSEE system was developed with major support from Dr. Hassan Gomaa, Dr. Larry Kerschberg, Dr. Richard Fairley, Chris Bosch, Vijayan Sugumaran, Iraj Tavakoli, and Elizabeth O'Hara-Schettino of the George mason University.

The KAPTUR system prototype was developed with major support from Dr. Sidney Bailin, Manju Bewtra, and Dick Bentz from CTA, Inc. Mike Moore, formerly of CTA but now with NASA/Goddard, also contributed to the KAPTUR development activity.

The success of the current systems is due to the creativity and hard work of these individuals.

REFERENCES

1. Gomaa, H., R. Fairley, L. Kerschberg, "An Evolutionary Domain Life Cycle for Software Maintenance", Report for NASA, 1991

2. Gomaa, H., L. Kerschberg, "An Evolutionary Domain Life Cycle for Domain Modeling and Target System Generation', Report for NASA, 1991

3. Bailin, S., R. Gattis, W. Truszkowski, "A Learning-Based Software Engineering Environment for Reusing Design Knowledge", Report for NASA, 1991

Copies af these reports are available from Walt Truszkowski